Reducing Software Developer Human Errors by Improving Situation Awareness

Bhaveet Nagaria
Department of Computer Science, Brunel University London

Tracy Hall
School of Computing and Communications, Lancaster University

***Abstract*—Software development is a human activity prone to human error. These errors are partially related to losing situation awareness during development tasks. Situation awareness enables the retention of contextual knowledge while performing a task. The OODA loop is an established cognitive training method to improve situation awareness. We studied the in-situ development errors that ten professional software developers made before and after using the OODA loop. Our preliminary results suggest that developer errors reduce after OODA loop use. We recommend that developers: get to know their own development weaknesses, use cognitive training (e.g., OODA loop) to manage those weaknesses, simplify their working environment and communicate carefully with external stakeholders.**

Introduction

Despite the tools [1] and processes [2] used during development, defects in software systems regularly occupy news headlines. We investigate whether defects can be reduced by focusing on the human errors made during development. We show that when ten developers used our online training to improve situation awareness, errors were reduced.

Human Error Theory [3] frames our study having recently attracted attention in software engineering [4], [5], [6], [7] and been previously used in other disciplines. James Reason [3] defines three types of human errors: slips, lapses and mistakes (Sidebar 1).

Errors are introduced during two phases of human cognition [3]; planning and execution. Mistakes occur during planning; slips and lapses during execution.

Sidebar 1: Human Error Types

Slip: Carelessness, e.g. entering commands in the wrong window.
Lapse: Forgetfulness, e.g. not saving a new build configuration.
Mistake: Unawareness, e.g. incorrectly describing a bug.

Maintaining situation awareness is an approach to reducing human errors successfully used in other domains (e.g. autonomous driving [8], medicine [9], transportation [10] and cyber security [11]). Situation awareness is maintaining an understanding of what is going on around you while performing a task [12]. Endsley describes three levels of situation awareness: (1) perception of the environment, (2) comprehension of the situation, (3) predicting the future situation. Agile software teams have improved their situation awareness using a 'project wall' to retain oversight of the whole project while developers perform individual tasks [13].

The OODA loop is a cognitive training method designed to improve decision-making [14]. The four stages of the OODA loop (Figure 1) encourages the maintenance of situation awareness (SA) by iteratively 'Observing' (level 1 of SA), 'Orienting' (level 2 of SA), and 'Deciding' before 'Acting'. We developed an online OODA loop training package to help developers maintain situation awareness during their development tasks (available at: https://bit.ly/2V6OOVL). This training package consists of four sections where developers learn about: situation awareness, the OODA loop, applications of the OODA loop within software development and then demonstrate their understanding through a nine question quiz.

We now discuss the experimental use of our training package by professional developers.

Method

Full methodological detail of our experimental approach is in our on-line appendix (https://bit.ly/2V6OOVL) of which we now provide a summary. We performed a 'before-and-after' OODA loop training experiment with ten professional software developers of whom we report findings from seven (two developers withdrew from the study part way through due to increased workloads, and one did not complete the training quiz). The developers had a variety of backgrounds and were a convenience sample. Each developer self-recorded the daily errors they made over five days of in-situ development activity. Each developer then undertook our OODA loop training, afterwhich developers recorded their errors over the next five days. Themes for all errors were independently coded by both authors with an initial agreement of 61.32%. Discussion of each disagreement (recommended by [15]) resulted in 100% Agreement. We identified seven themes across the human errors (see Sidebar 2).

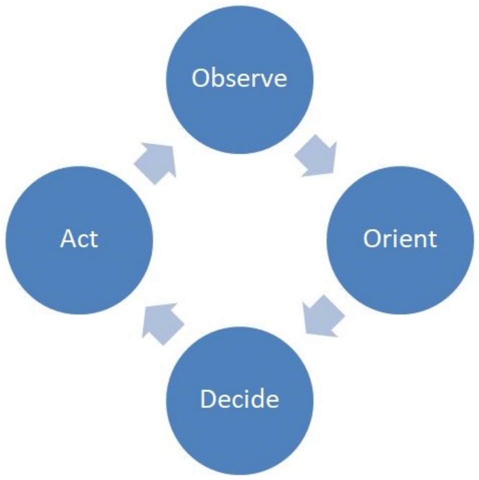

Figure 1. OODA Loop

Results

Our small-scale snapshot study confirms that developers make human errors manifesting as slips, lapses and mistakes. Training software developers to maintain situation awareness using the OODA Loop seems to lead to decreased developer errors. We now discuss our findings in more detail.

What type of human errors do developers make?

Table 1 shows each error for each developer classified two ways; first, errors in each of the seven themes, second, each error classified as either a slip, lapse or mistake. A small number of errors are multi-classified because they cut across human error classifications.

Table 1 suggests that the distribution of human error types (i.e. slips, lapses and mistakes) varies between developers with some reporting many more mistakes than others.

Sidebar 2: Human Error Themes

Internal communication Poor internal communication e.g. incomplete documentation.
External communication Poor external communication e.g. failing to obtain full error details from end user.
Code structure/complexity Poor code structure and or increased code complexity.
Complexity of development environment Having many things running in the development environment at anyone time.
Ordering/sequencing tasks Executing a series of tasks in the incorrect order.
Syntax issues Use the wrong syntax / syntax errors in newly written code.
Special cases Unique errors which do not fit the other themes e.g. UI/UX design / functionality issues

Table 1 also shows variation in the error themes to which developers seem prone. For example, Participant 6 makes the most syntax errors, a number of which appear to be related to JavaScript and specifically the use of the keyword 'this'. Whereas Participant 8 seems to make more code structure / complexity errors. Participant 5 is the only developer who does not record any communication related errors. Our analysis (see online appendix for further details) does not suggest experience or demographics explains why these developers seem prone to these specific errors but it might be that the particular work tasks during the snapshot influenced the error themes. Future work is needed to investigate the relationship between errors and development context.

Table 1 suggests that most communication based errors are mistakes rather than slips or lapses. Mistakes are usually more substantial errors than lapses or slips and can be more complex to correct. This confirms the importance of strong communication during development activities. Table 1 also shows that syntax errors comprise mostly of slips or lapses. This suggests that developers generally know syntax but make minor errors despite this knowledge. Most developers reported errors related to the complexity of the development environment. For example Participant 10 says that they *'Forgot to increase the version of an updated dependency.'* This is because Participant 10 says they were *'Juggling three different tasks all at the same time. Performance research on one strand, bug fixing on two separate issues. Each with their own programming languages! (Python, Java and Scala)'*. Table 1 shows only Participants 8 and 9 do not report such complex development environment errors. More work is needed to uncover any contextual factors explaining this variation.

Does OODA loop training lead to a reduction in human errors?

Developers engaged well with our OODA loop training. All but one of whom attained scores of eight or nine in the nine question quiz that concluded the training package.

Table 2 shows how many errors were self-recorded by developers before and after OODA loop training. Although the numbers of errors in the snapshot are small, Table 2 shows an encouraging error reduction after training. In all cases there is either a reduction in errors (four developers) or no change in the number of errors (three developers) after the OODA loop training. We perform a paired T-test, which shows a significant difference of 0.0414 between the number of errors being made before and after the OODA loop training.

We looked in more detail at the types of errors before and after OODA loop training and found that error reductions after training were predominately in execution errors. For example Participant 6 makes 18 execution errors before training which reduces to four execution errors after training. This reduction in execution errors suggests that using the OODA loop during development helps developers retain situation awareness of their code and maintain concentration sufficiently to reduce the slips and lapses that they usually make. A larger scale study is needed to establish whether this finding holds more generally for developers and whether the effect lasts over time.

Do developers find OODA loop training useful?

We asked developers for their thoughts about the training. The sentiment of all responses is positive indicating that participants enjoyed par-

Table 1. High Level Themes

	Team Communication						
Participant Number	**Internal**	**External**	**Code Structure / Complexity**	**Complexity of Development Environment**	**Ordering / Sequencing Tasks**	**Syntax Issues**	**Special Cases**
P1		M1	L/M 1	S 1	S/L/M 1 S 1 L 1	S/L 1	
P5				S 1 S 2 S 3	L 1 S/L/M 1		S/L 1
P6	S/L/M 1	M 1 M 2 M 3	S/L 1 S/L 2 S/L 3	S/L 1		S/L 1 S/L 2 S/L 3 S/L 4 S/L 5 S/L 6	
P7	M 1	M 1 S/L/M 1 S/L/M 2	S/L 1	S/L 1 S/L/M 1 M 1		S/L 1 S/L 2 S/L 3 M 1 L 1	S/L 1
P8	M 1		S/L 1 S/L 2 S/L 3 S/M 1 S/L 4				
P9	M 1	M 1 M 2	S/L/M 1 S/L 2			S/L 1 S/L 2	M 1
P10		M 1	L/M 1	L 1 M 1 L 2	S 1 L 1	S 1 L 1 S 2	

Day 1 to 5: White Background & Day 6 to 10: Grey Background
Key: S = Slip, L = Lapse, M = Mistake, S/L = Slip/Lapse, L/M = Lapse/Mistake & S/L/M = Slip/Lapse/Mistake
Note: The number after the key is a counter for each participant for the type of error
Note: Participants 2, 3, 4 withdrew due to a change in workload or non - engagement

Table 2. Numbers of logged Human Errors

	Total Human Errors	**Before Training Errors: D1- D5**	**After Training Errors: D6 - D10**	**Reduction Rate**
P1	7	6	1	83%
P5	6	3	3	0%
P6	14	9	5	44%
P7	14	8	6	25%
P8	6	5	1	80%
P9	8	4	4	0%
P10	10	5	5	0%

ticipating in the study and found the online training tool easy to use. Participants said that they learned about the OODA loop and how to apply it in software development for the first time. Participant 7 said *'Yes, the idea of OODA was helpful when dealing with developing problems.'* All developers found the content actionable, Participant 2 saying *'Fully actionable especially in our work.'* Participant 9 says the OODA loop is *'Very helpful and is a need in our daily work in software engineering.'* While the majority of the participants found the OODA loop led to an improvement in their work, Participants 1 & 10 did not. All except Participant 1 report that they will continue to use the OODA loop in the their work. Participant 10 says that they will continue to use the OODA loop in their development, despite not finding any direct improvement.

Recommendations for reducing developer errors

Our preliminary findings lead to the following recommendations to help software developers reduce the errors they make.

- **Know your own weaknesses.** Every developer is different and struggles with different concepts. Our analysis shows a variety of types of errors that developers make. Developers becoming more conscious of the human errors they commonly make and actively checking for these can help reduce errors.
- **Use cognitive training.** We have shown that using cognitive training, like the OODA loop, seems to help decision making and can reduce

the human errors a developer makes.

- **Simplify your workload.** One of the biggest causes of human error reported by the developers in our study was the complexity of the development environment. Reducing the cognitive load by simplifying the complexity of the development environment could reduce human errors. Actions such as minimising the number of simultaneous development tasks and closing down unnecessary tools and windows can help reduce the cognitive load.
- **Communicate carefully with stakeholders outside your team.** Our study suggests that a relatively high number of mistakes are related to communicating with stakeholders outside of the development team. Ensuring that communication is clearly understood seems important to reducing mistakes.

Conclusion

We presented our preliminary study on reducing human error during development by providing developers with cognitive training designed to improve their situation awareness. Our study makes the following contributions:

- To the best of our knowledge this is the first study to provide empirical evidence showing how the number of human errors made by professional software developers reduces when developers are provided with training to improve their situation awareness. More work is needed to understand whether our results hold beyond this small scale study.
- We show that software developers do make all three types of human errors (slips, lapses and mistakes) in their development work. Our results also suggest that slips and lapses occur more commonly than mistakes. Slips and lapses are likely to result in smaller defects which should be easier to mitigate.
- We show that cognitive training using the OODA loop leads to a decrease in the number of human errors software developers make. We found that developers particularly reduced the number of execution errors they made following OODA loop training. More work is needed to understand if this finding is generalisable and if the effect lasts over time.

We were encouraged that most of the developers in our study were enthusiastic about the training they received in how to maintain their situation awareness. Most developers found the training easy and useful in a topic they had no previous knowledge. The majority of the developers said that they will continue to use the OODA loop in their work. We intend to explore further the use of focused training packages to educate and support developers in their work.

Acknowledgments

The authors would like to thank all the study participants. This study has has been approved by Brunel University Ethics Committee reference 18067-LR-Oct/2019-20590-1. The study was partly funded by the UK's Engineering and Physical Sciences Research Council (EP/S005730/1).

REFERENCES

1. F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. Di Penta, "How open source projects use static code analysis tools in continuous integration pipelines," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 334–344.
2. S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: a case study of the qt, VTK, and ITK projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 192–201.
3. J. Reason, *Human Error*. New York; Cambridge [England]: Cambridge University Press, 1990.
4. V. Anu, G. Walia, W. Hu, J. Carver, and G. Bradshaw, "Effectiveness of human error taxonomy during requirements inspection: An empirical investigation," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, vol. 2016-Janua, 2016, pp. 531–536.
5. F. Huang, "Post-completion Error in Software Development," *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering*, pp. 108–113, 2016.
6. W. Hu, J. C. Carver, V. Anu, G. Walia, and G. Bradshaw, "Defect Prevention in Requirements Using Human Error Information: An Empirical Study," pp. 61–76, 2017.
7. F. Huang and B. Liu, "Software defect prevention based

on human error theories," *Chinese Journal of Aeronautics*, 2017.

8. L. Petersen, L. Robert, J. Yang, and D. Tilbury, "Situational Awareness, Driver's Trust in Automated Driving Systems and Secondary Task Performance," *SAE International Journal of Connected and Autonomous Vehicles, Forthcoming*, 2019.
9. M. C. Wright, J. M. Taekman, and M. R. Endsley, "Objective measures of situation awareness in a simulated medical environment," *BMJ Quality & Safety*, vol. 13, no. suppl 1, pp. i65–i71, 2004.
10. C. D. Wickens, "Situation awareness and workload in aviation," *Current directions in psychological science*, vol. 11, no. 4, pp. 128–133, 2002.
11. G. Ioannou, P. Louvieris, and N. Clewley, "A Markov Multi-phase Transferable Belief Model for Cyber Situational Awareness," *IEEE Access*, 2019.
12. M. R. Endsley, "Situation awareness global assessment technique (sagat)," in *Proceedings of the IEEE 1988 national aerospace and electronics conference*. IEEE, 1988, pp. 789–795.
13. J. Srinivasan and K. Lundqvist, "Using agile methods in software product development: A case study," in *2009 Sixth International Conference on Information Technology: New Generations*. IEEE, 2009, pp. 1415–1420.
14. J. Boyd, "A discourse on winning and losing [briefing slides]," *Maxwell Air Force Base, AL: Air University Library.(Document No. MU 43947)*, 1987.
15. B. Kitchenham, D. I. Sjøberg, T. Dybå, O. P. Brereton, D. Budgen, M. Höst, and P. Runeson, "Trends in the quality of human-centric software engineering experiments–a quasi-experiment," *IEEE Transactions on Software Engineering*, vol. 39, no. 7, pp. 1002–1017, 2012.

Bhaveet Nagaria is working toward the PhD degree with Brunel University London. His research interests include human factors and software engineering. Contact him at bhaveet.nagaria@brunel.ac.uk; http://www.brunel.ac.uk/bhaveet-nagaria

Tracy Hall is Professor of Software Engineering at Lancaster University. Her research interests include code analysis, defect prediction and human factors of developers. Contact her at tracy.hall@lancaster.ac.uk; https://www.lancaster.ac.uk/scc/about-us/people/tracy-hall